Physical Design Automation for VLSI and FPGAs

Routing

Mohammed Khalid

Department of Electrical and Computer Engineering

University of Windsor

References and Copyright

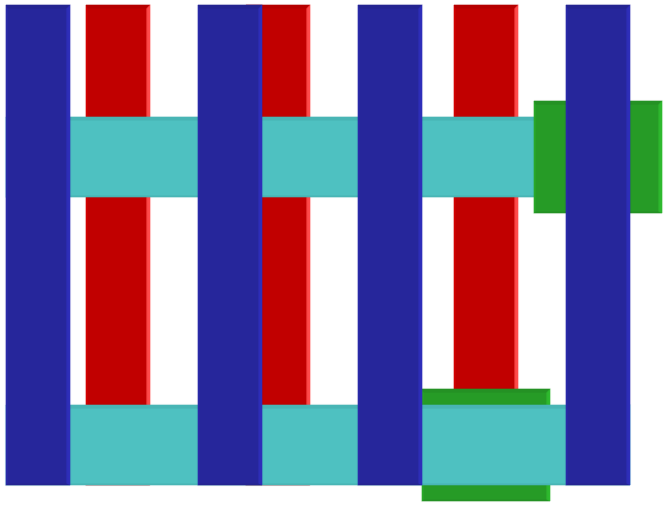
- Slides sources (modified by Khalid as needed):
  - Prof. Kia Bazargan, Univ. of Minnesota
  - Prof. Majid Sarrafzadeh, UCLA
  - Dr. Naveed Sherwani,(companion slides with textbook)
  - Prof. Kurt Keutzer, UC-Berekeley
  - Prof. Rajesh Gupta, UC-Irvine
  - Prof. Steve Kang, Univ. of Illinois (Urbana)
  - Prof. Jonathan Rose (Univ. of Toronto)

Routing

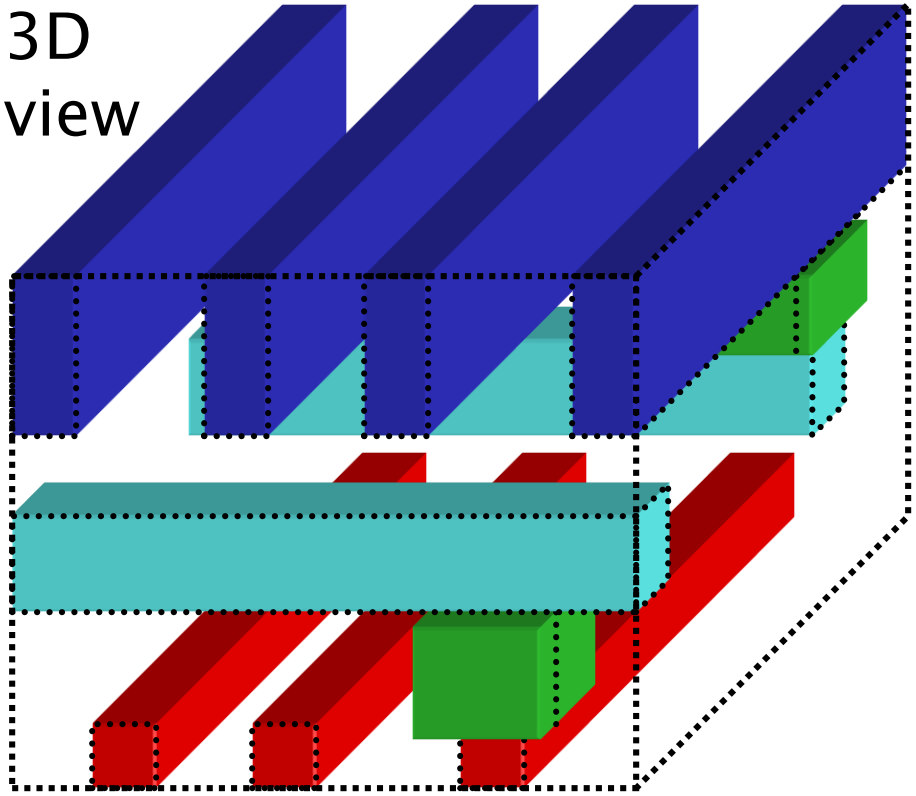
- Problem
  - Given a placement, and a fixed number of metal layers, find a valid pattern of horizontal and vertical wires that connect the terminals of the nets
  - Levels of abstraction:
    - Global routing
    - Detailed routing
- Objectives
  - Cost components (to minimize):
    - Area (channel width) - minimize congestion
    - Wire delays – minimize wire length
    - Number of layers (less layers → less expensive)
    - Additional cost components: number of bends, vias

Routing Anatomy

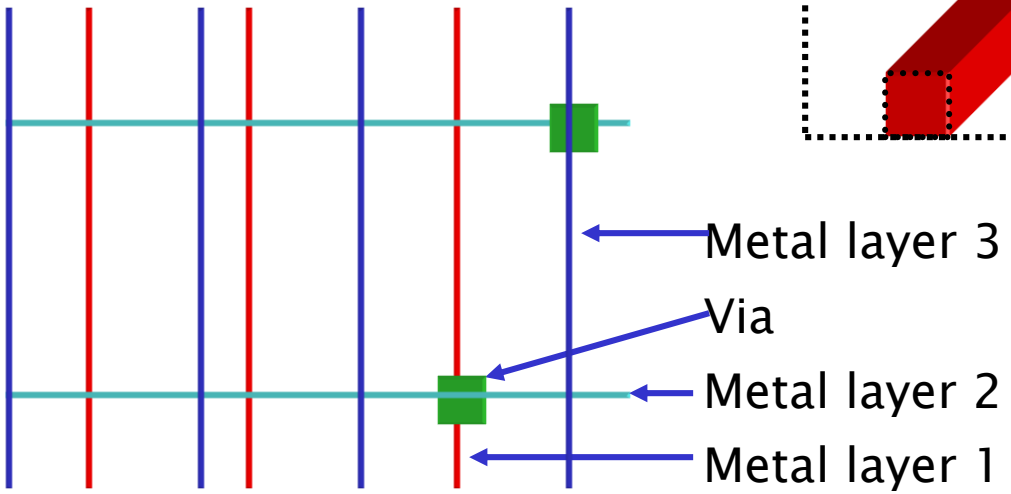
Top view



3D view



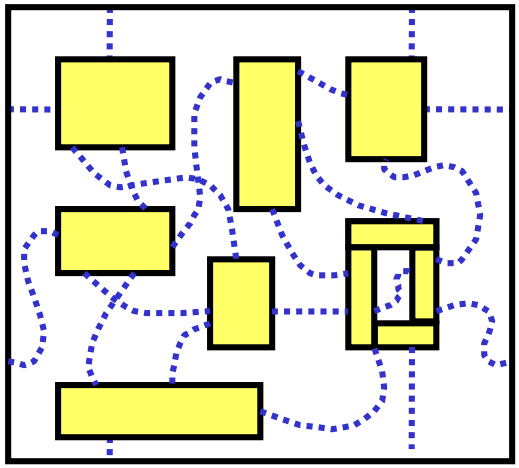
Symbolic Layout



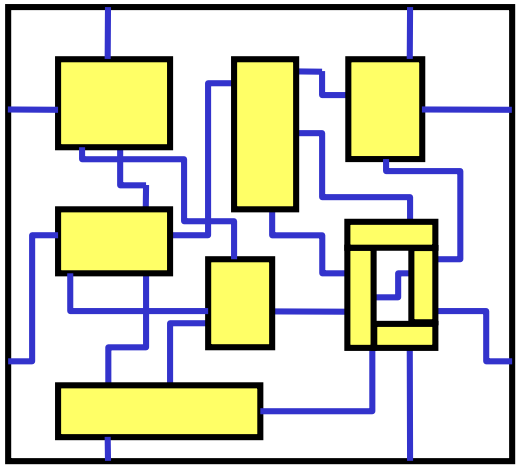
Note: Colors used in this slide are not standard

Global vs. Detailed Routing

- Global routing
  - Input: detailed placement, with exact terminal locations
  - Determine "channel" (routing region) for each net
  - Objective: minimize area (congestion), and timing (approximate)



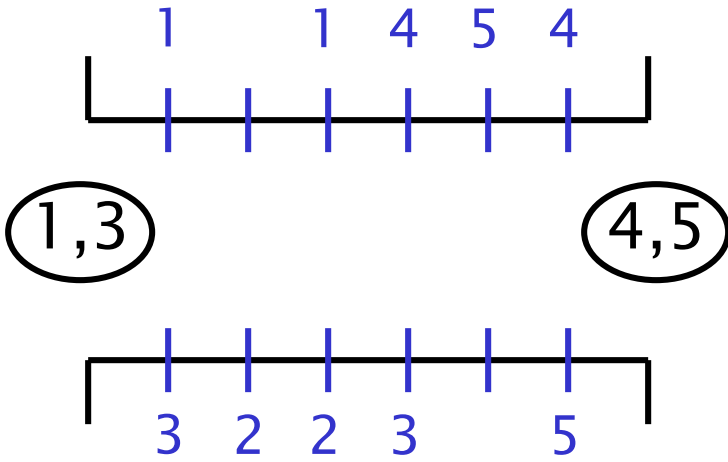
- Detailed routing
  - Input: channels and approximate routing from the global routing phase
  - Determine the exact route and layers for each net
  - Objective: valid routing, minimize area (congestion), meet timing constraints
  - Additional objectives: min via, power



Figs. [©Sherwani]

Routing Environment

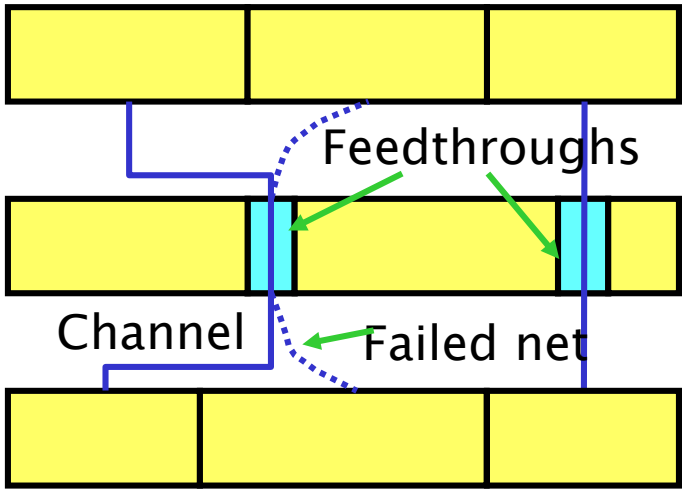
- Routing regions
  - Channel
    - Fixed height ? (→ fixed number of tracks)
    - Fixed terminals on top and bottom
    - More constrained problem: switchbox. Terminals on four sides fixed

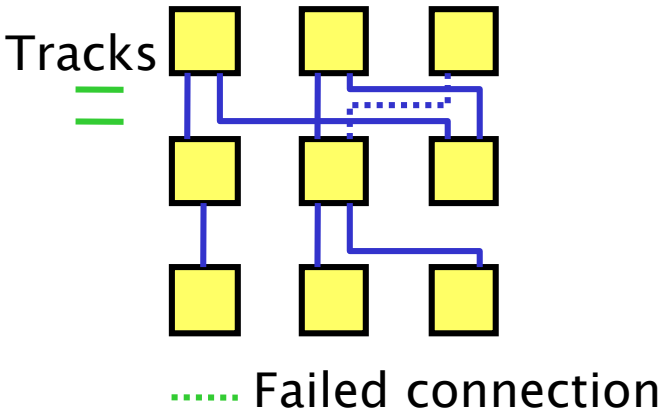


  - Area routing
    - Wires can pass through any region not occupied by cells (exception: over-the-cell routing)
- Routing layers
  - Could be pre-assigned (e.g., M1 horizontal, M2 vert.)
  - Different weights might be assigned to layers

Routing Environment

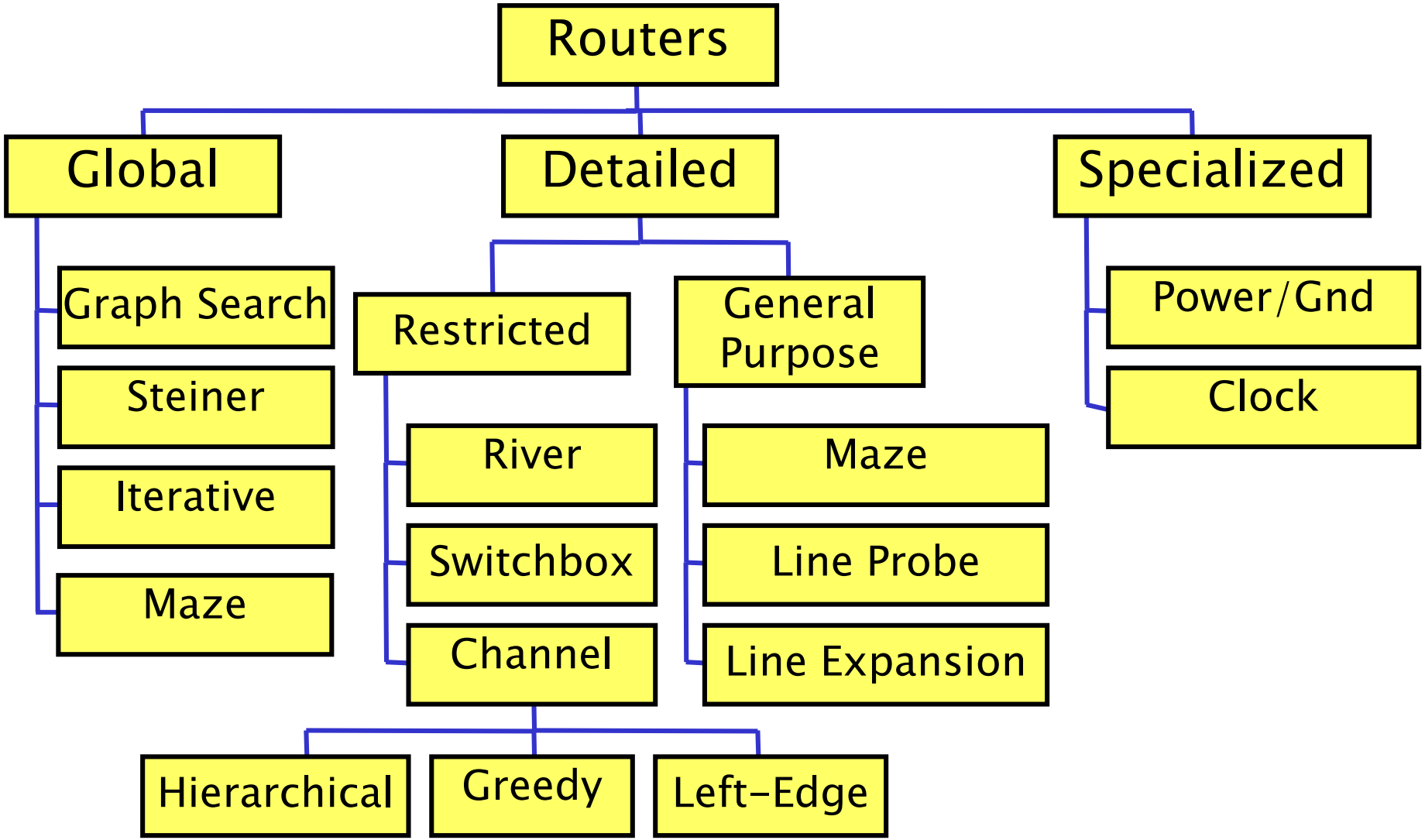
- Chip architecture
  - Full-custom:
    - No constraint on routing regions
  - Standard cell:
    - Variable channel height?
    - Feed-through cells connect channels
  - FPGA:
    - Fixed channel height
    - Limited switchbox connections
    - Prefabricated wire segments have different weights





Figs. [©Sherwani]

Taxonomy of VLSI Routers



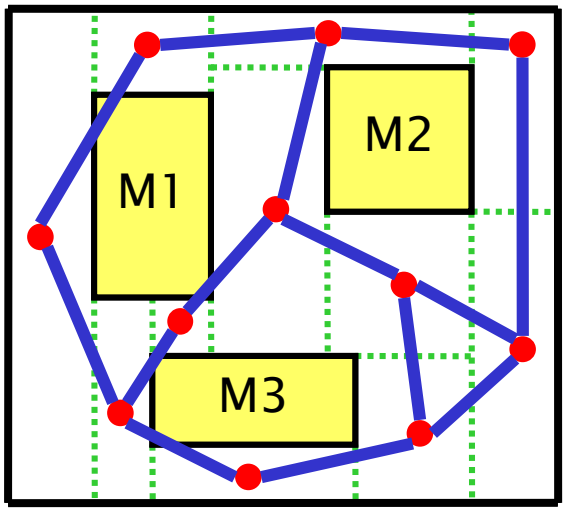
[©Keutzer]

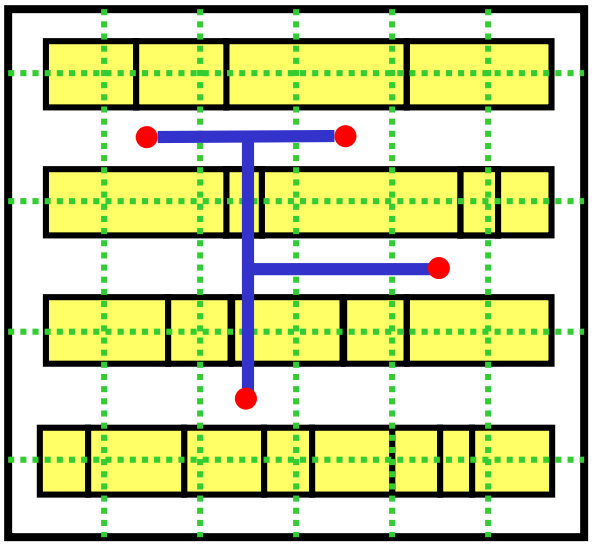
Global Routing

[©Sarrafzadeh]

- Stages
  - Routing region definition
  - Routing region ordering
  - Steiner-tree / area routing
- Grid
  - Tiles super-imposed on placement
  - Regular or irregular
  - Smaller problem to solve, higher level of abstraction
  - Terminals at center of grid tiles
- Edge capacity
  - Number of nets that can pass a certain grid edge (aka congestion)
  - On edge E_{ij},

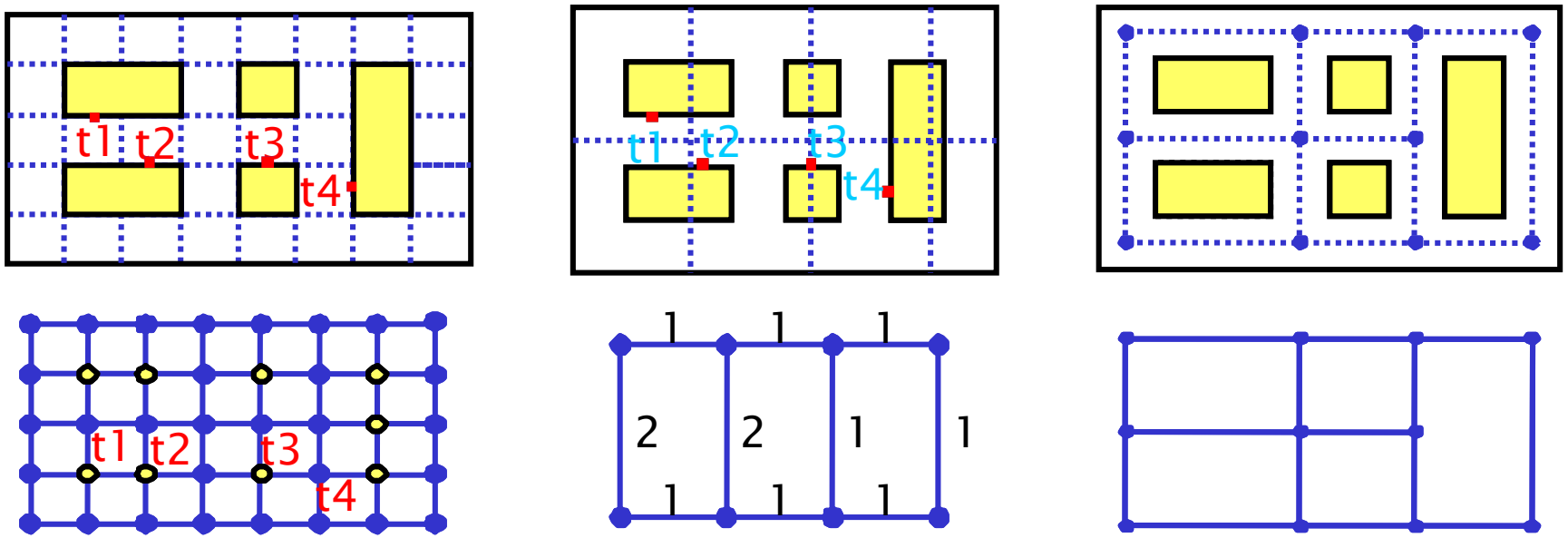
$$Capacity(E_{ij}) \geq Congestion(E_{ij})$$





Grid Graph

- Course or fine-grain
- Vertices: routing regions, edges: route exists?
- Weights on edges
  - How costly is to use that edge
  - Could vary during the routing (e.g., for congestion)
  - Horizontal / vertical might have different weights



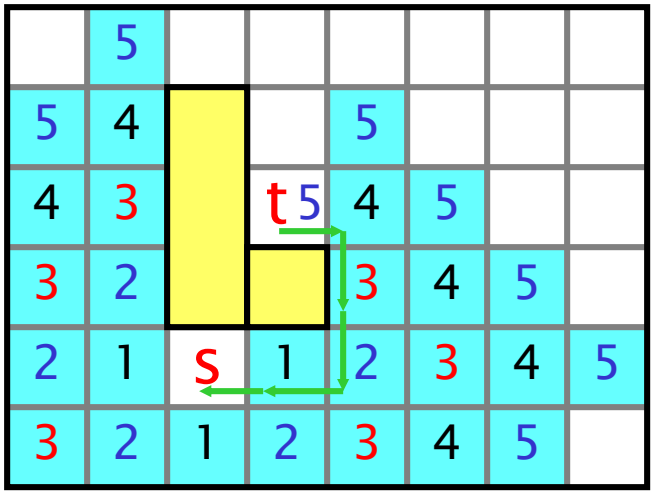
[©Sherwani]

Global Routing – Graph Search

- Good for two-terminal nets
- Build grid graph (Coarse? Fine?)
- Use graph search algorithms, e.g., Dijkstra
- Iterative: route nets one by one
- How to handle:
  - Congestion?
  - Critical nets?
- Order of the nets to route?
  - Net criticality
  - Half-perimeter of the bounding box
  - Number of terminals

Global Routing – Maze Routing

- Similar to breadth-first search
  - Very simple algorithm
  - Works on grid graph
  - Time complexity: grid size (NxN)
- Algorithm
  - Propagate a "wave" from source until it hits the sink (implemented using a queue)
  - Trace back to find the path



- Guaranteed to find the optimal solution
  - Usually multiple optimal solutions exist
- More than two terminals?
  - For the third terminal, use the path between the first two as the source of the wave

Maze Routing

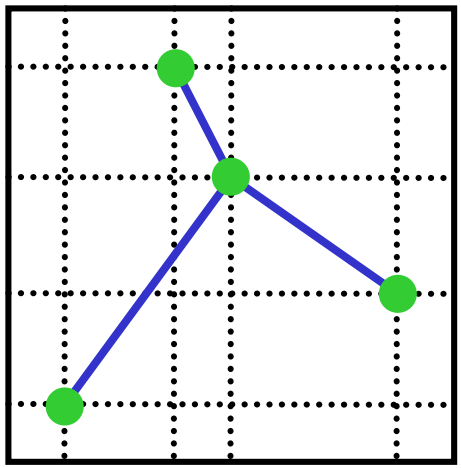
- Key to popularity:
  - Simplicity
  - Guaranteed to find the optimal solution if it exists
  - Can realize more complex cost functions too (e.g., number of bends in a path)
- Weakness:
  - Multiple terminals not handled efficiently
  - Dependent on grid, a two dimensional data structure
- Different variations exist
  - Soukup's alg:
    - First use DFS, when get to an obstacle, use BFS to get around
    - No guarantee to find the shortest path

Multiple Terminal Nets: Steiner Tree

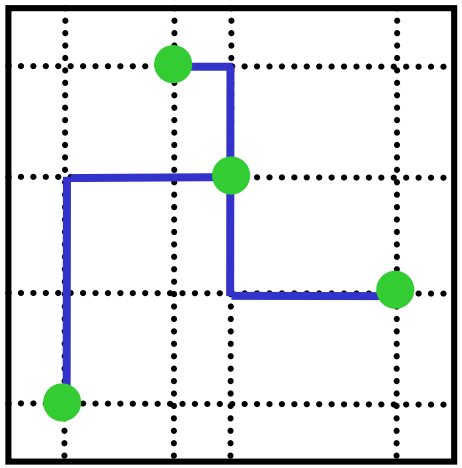
- Steiner tree (aka Rectilinear Steiner Tree – RST):
  - A tree connecting multiple terminals
    - Original points: "demand points" – set D
    - Added points: "Steiner points" – set S
  - Edges horizontal or vertical only
- Steiner Minimum Tree (SMT)
  - Similar to minimum spanning tree (MST)
  - But finding SMT is NP-complete
  - Many good heuristics introduced to find SMT
- Algorithm
  - Find MST
  - Pass horizontal and vertical lines from each terminal to get the Hannan grid (optimal solution is on this grid)
  - Convert each edge of the MST to an L-shaped route on Hannan grid (add a Steiner point at the corner of L)

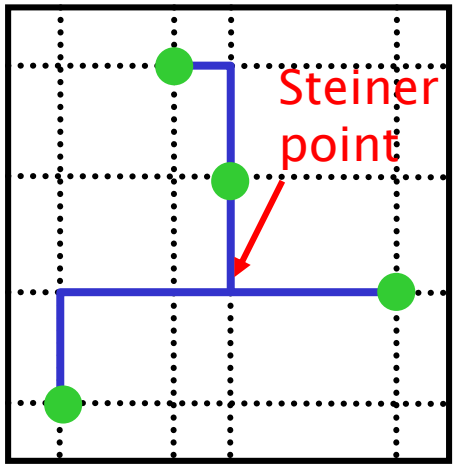
Steiner Tree

- Hannan grid reduces solution space (smaller grid)
  - For min length RST, Steiner points always on Hannan grid
- Convert MST to rectilinear paths
  - Length bounded by 1.5 times optimal SMT length
- Use alternate "L" routes to find the minimum tree



MSP (length=11)





Steiner tree (len=13)

[©Sherwani]

Steiner Tree Routing

- Can apply different costs to different regions (or horizontal/vertical preference)
- Order of the nets
  - Sequential
    - Use # of terminals, criticality, etc. to determine order
  - Parallel
    - Divide the chip into large regions, perform the routing in parallel
- Key to popularity
  - Fast (not theoretically, but practically)
  - Bounded solution quality
- Shortcomings
  - Difficult to predict or avoid congestion

Global Routing for Standard Cells

- Standard cells are usually used for random logic, FSMs and small logic and functional blocks
- Well developed CAD tools for layout, highly effective synthesis tools
- Designer/logic synthesis tool creates desired circuit from pre-designed cells - AND2, NOR3, DFF, etc.
- Easy to layout: (1) fixed height, variable width (2) Power rails at same fixed position on each cell, can 'abut' cells together (3) Physical pin for each logical cell pin available on both top and bottom of each cell - key point for routing (example of AND layout) (4) routing done in channels between rows (5) use feedthrough cells for inter-channel connections (to cross a row of cells)

Global Routing for Std. Cells: Problem Def.

- **Given**: (1) Placement of cells in rows (2) netlist of **logical** pin connections
- **Find**: (1) What channels to use for routing each net (2) where feedthroughs (if needed) are placed for routing each net (3) which **physical** pins to connect for routing each net
- Optimization goal: Minimize $\sum(Channel\_Widths)$
  - which roughly means minimize $\sum(Channel\_Densities)$
- Keep an eye on number of feedthroughs used --> will increase row width and possibly area
  - avoid heavily used channel sections by clever choices of where to put feedthroughs (see Fig. 2.1 and 2.2)

Global Routing - Cong/Preas Algorithm

- Think about all connections at the same time - concurrent routing algorithm, as opposed to iterative routing (net by net)
- Define a net connection graph , NCG, G(P, E), where P is set of nodes (vertices) and E is a set of edges.
  - P is a set of all logical pins on all nets
  - there is an edge between two nodes (Pi, Pj) if they are on the same net
  - Example: Assuming P1, P2, P3, P4 are logical pins of one net (see figure in notes)
  - feedthrough will add a node in graph
- Have one NCG for each net. For best routing construct minimum spanning tree (MST) for each net
- For complete routing, we need a collection of MSTs (one for each net), i.e. a minimum spanning forest

Cong/Preas Algorithm

Step 1: Choose feedthrough locations & edges (Sec. 3.1)

- Choosing an edge that spans across a row => choosing an associated feedthrough location (see figure in notes)
- For each pin Pi (Xi, Yi), Xi denotes horizontal position, Yi denotes vertical position
- Cost of an edge from Pi to Pj is:

  | Xi - Xj | + K * {feedthrough cost in each traversed row from Yi to Yj}

  - second term depends on whether longest row is made longer
  - K is a constant that is experimentally determined
  - cost of edge is a function of both wire length & cost of adding feedthroughs
- Calculate edge cost for all edges (feedthrough costs change as edges are selected, hence edge costs change as well)

Cong/Preas Algorithm

Algorithm 3.1: Selection of edges (determining feedthroughs)

Let set *E* contain all edges in forest (set of NCGs, one for each net)
Let set *F* contain the minimum spanning forest (empty initially, i.e. *F* = ø)

While (*E* <> ø)
{
  remove the minimum weighted edge *e* from *E;*
  if (*F U e*) does not have a cycle
    then include *e* in *F;* recalculate edge costs in *E* if feedthroughs added;
  // if edge creates a cycle, throw it away because it is useless
}
// Now *F* contains the minimum spanning forest (with feedthroughs)

- We consider edges of all nets at the same time => algorithm is **independent** of net order

Cong/Preas Algorithm

Step 2: Determine net segments (choose physical pins)

- Now we have a way to connect into every channel of a net
- Don't need to cross rows (feedthroughs already in place)
- set up graph as before, except that not all pins are connected --> just connect pins in channels that are neighbours

- At this stage chip width is fixed (by longest row) -- minimize channel widths to minimize chip height

Definition: simplified net connection graph (SNCG)

- Vertex set is the set of all logical pins in the design
- For two pins Pi and Pj, (Pi, Pj) is an edge iff the following is true:
  - Pi & Pj are on the same cell or feedthrough & are connected internally
  - Pi & Pj are in same channel & have no other pins of same net between them (see Figure 3-1)

Cong/Preas Algorithm

Definition: Weight of an edge *w (e) = d(e) / d*, where *d(e)* is the maximum density over the edge in the channel to which *e* belongs, & *d* is channel density

- w(e) is a measure of how close the edge is to contribute to channel density, w(e) =1 implies edge **does** contribute (see figure in notes)

Cong/Preas Algorithm

Algorithm 3.2: determine net segments

Build simplified net connection graph (SNCG);

S = Edge set of SNCG;

repeat

{

Remove maximum weighted edge in S on a cycle;

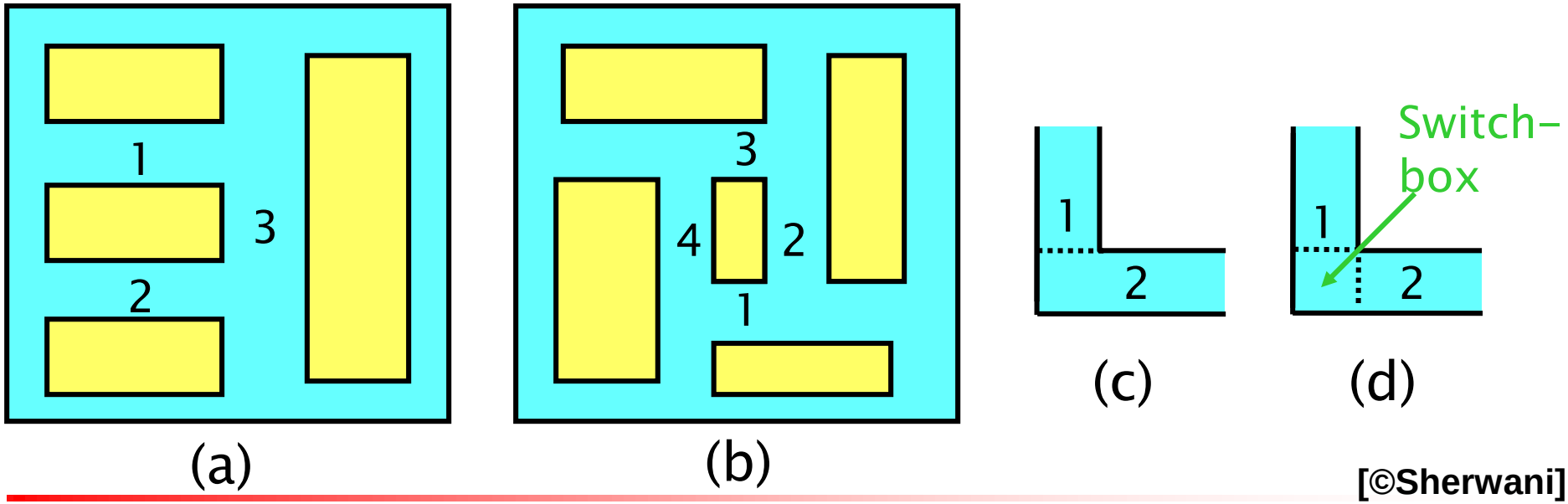
Update edge weights [w(e)] for affected edges;

} **until** S is a spanning forest // no cycle left in SNCG

- Since all edges (nets) are considered from the start, congestion is minimized from a global perspective -- as opposed to iterative net by net routing
- Time complexity of whole algorithm is ***O(n² * max(p, log[n])***
  - where n is number of physical pins and p is average of physical pins per net (average fanout over all nets)
- Results on benchmark circuits better than previous routers (Sec. 5)

Detailed Routing: Channel vs. Switchbox

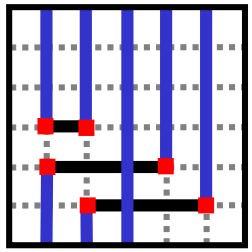
a) Channels have no conflicts
b) Conflicting channels
c) Conflict resolved using L-shaped channels
   - Order matters
d) Switchbox used to resolve the conflict
   - Order matters
   - Harder problem (compared to channel routing)



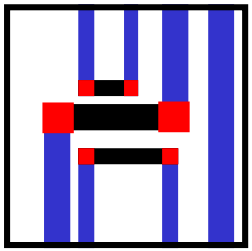
[©Sherwani]

Channel Routing Problem

- Input
  - Fixed terminal locations on the top and bottom
  - Possibly floating terminals on the left and right
  - Possibly fixed channel capacity constraint (capacity = max # of horizontal wires between the top and bottom boundaries of the channel)
  - Either gridless (aka area-based) or grid-based
- In the algorithms we consider
  - Constraints:
    - Grid structure
    - Two routing layers (one for H, another for V) Still relevant?
  - Minimize
    - # tracks (channel height)
    - Total wire length
    - # vias



Grid-based

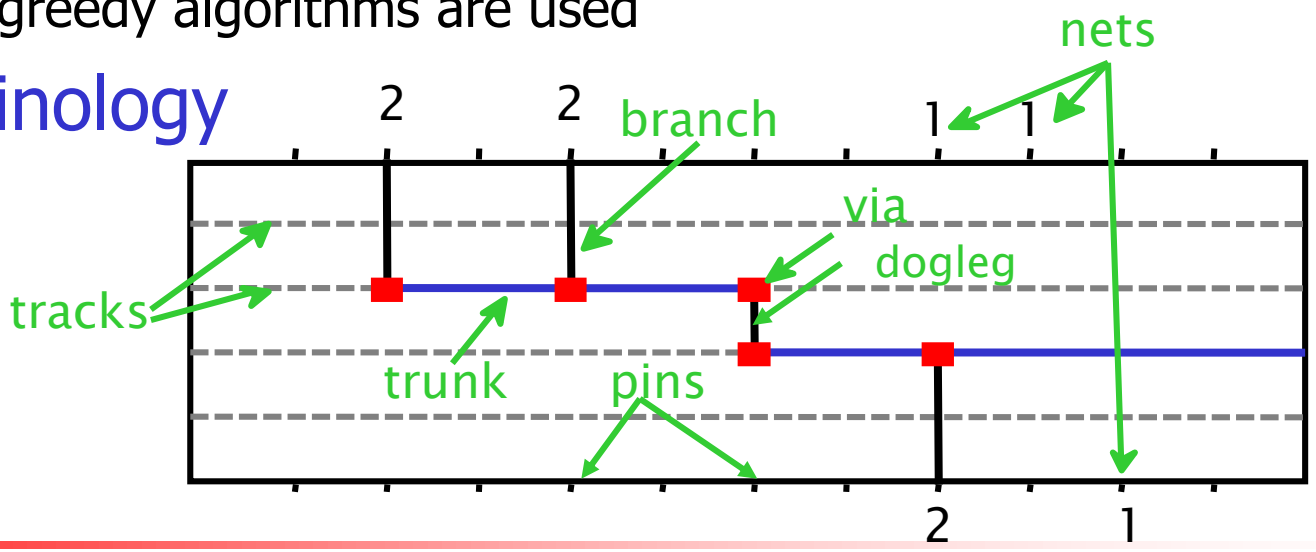


Gridless

[©Sherwani]
[©Keutzer]

Channel Routing Algorithms and Terminology

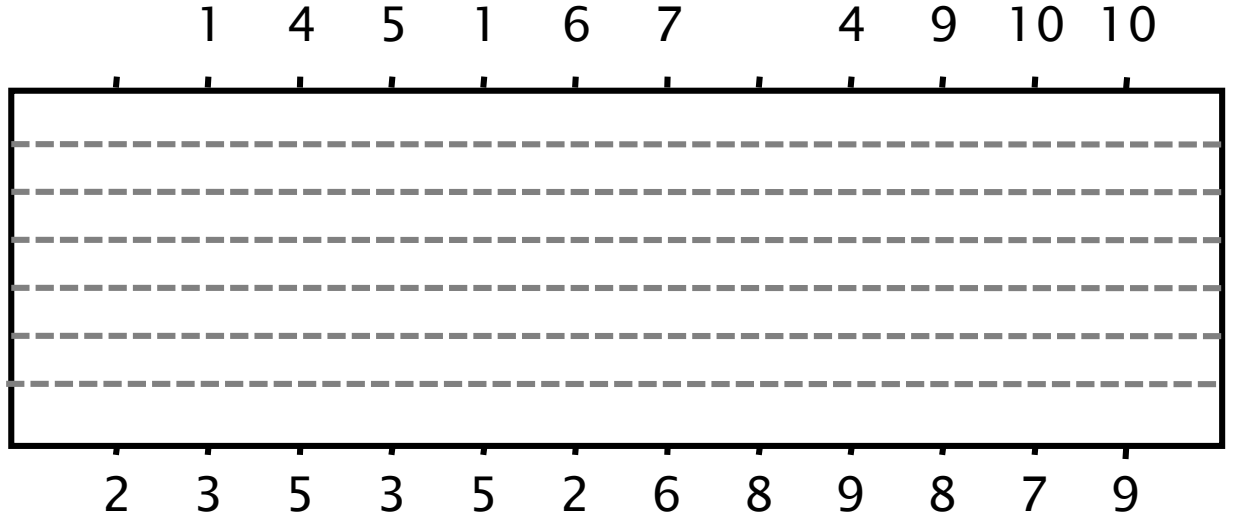
- General case is NP-Complete
- Algorithms
  - Simple case: left-edge algorithm (P)
  - General case: NP ➔ heuristics
    - Problem defined using horizontal constraints graph and vertical constraints graph
    - Either graph-based algorithms or greedy algorithms are used
- Terminology



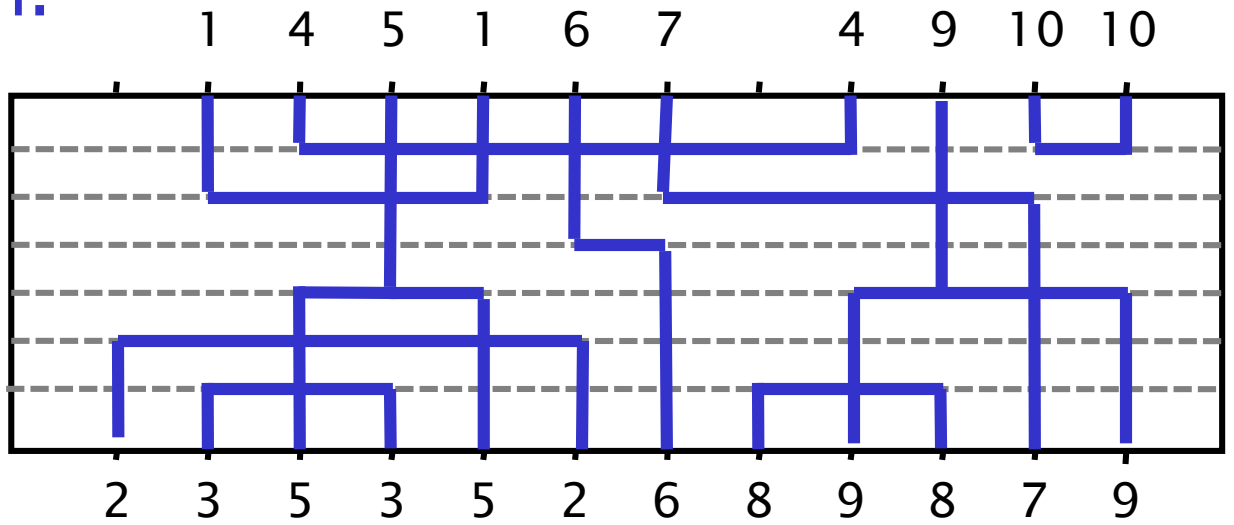
[©Keutzer]

Channel Routing Example

- Problem instance: (vacant terminals may be marked 0)



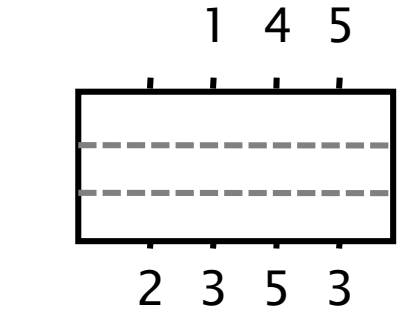
- Solution:

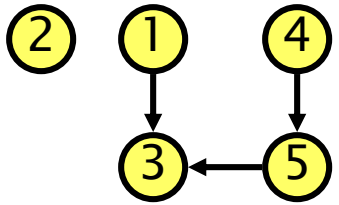


[©Keutzer]

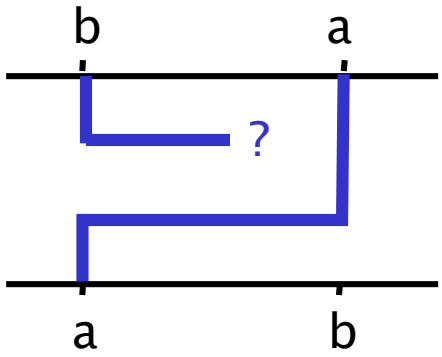
Vertical Constraint Graph

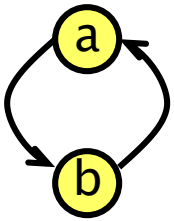
- Represents the relative vertical positions of different net trunks (tracks)
  - Node: represents a net
  - Edge (x, y): if at the same column, x has a terminal on the upper edge and y has a terminal on the lower edge
  - x is not equal to y
  - (a,b) means that net "a" has to be above "b"





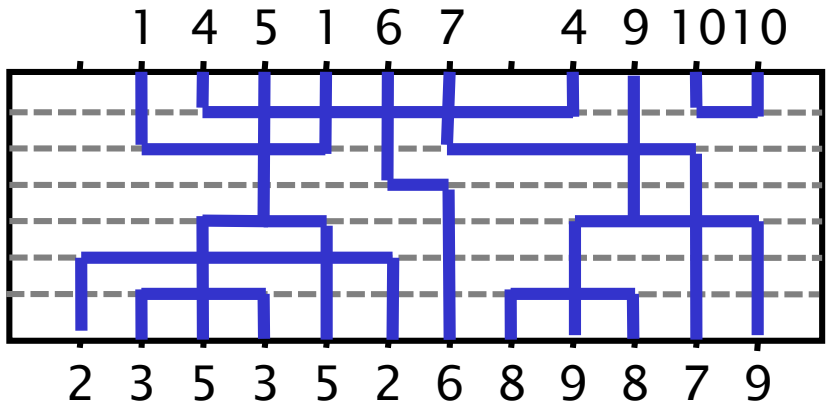
- Lower bound (without doglegs):
  - # tracks ≥ longest path in VCG
- VCG may have a cycle!
  - What shall we do? (more later)

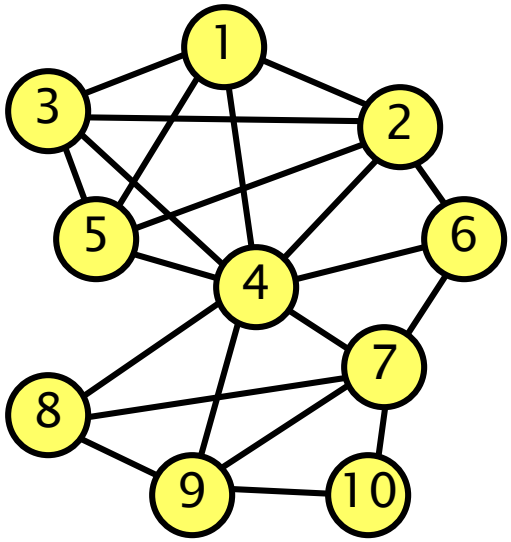




Horizontal Constraint Graph

- HCG is an undirected graph
- Each node is a net
- There is an edge between two nodes if the trunks (tracks) of these two nets overlap -- horizontal spans of nets overlap
- Maximal clique in HCG forms gives channel density (lower bound on number of tracks needed for routing)
- Lower bound is ***maximum*** of longest path in VCG or maximal clique in HCG



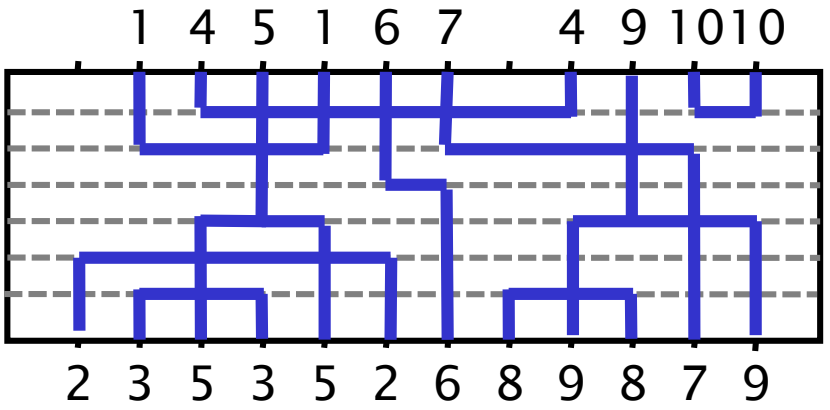


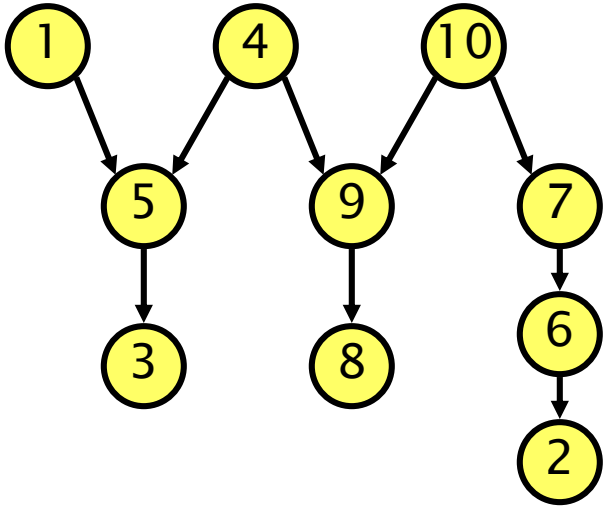
Horizontal constraint graph (HCG)

[©Keutzer]

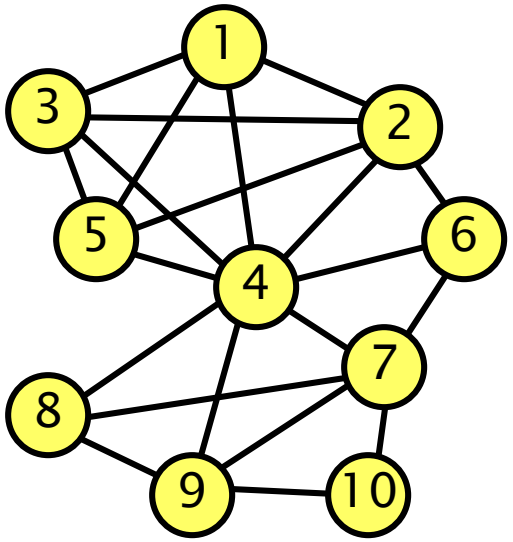
Horizontal / Vertical Constraint Graphs

- Channel routing problem can be completely characterized by (directed) VCG and (undirected) HCG





Vertical constraint graph (VCG)

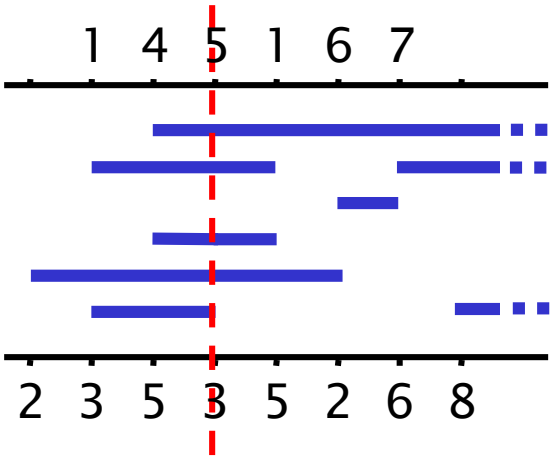


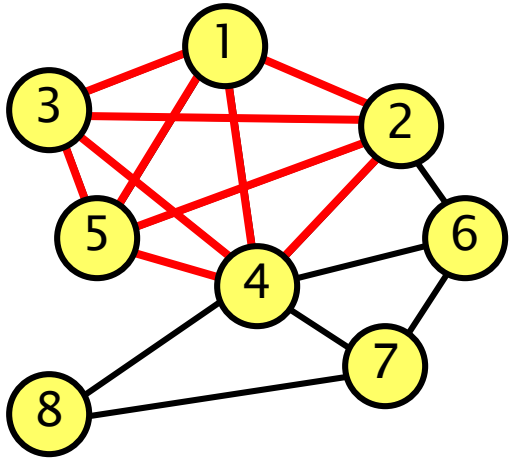
Horizontal constraint graph (HCG)

[©Keutzer]

Channel Density

- A net extends from its leftmost terminal to its rightmost one
- Local density at column C
  - ld (C) = # nets split by column C
- Channel density
  - d = max ld(c) over all C
- Relationship to HCG?
  - Local density ⇔ clique in HCG
  - d ⇔ size of maximum clique in HCG
- Lower bound:
  - # tracks ≥ d





[©Keutzer]

Left-edge Channel Routing Algorithm

- Assume no cycles in VCG & no doglegs allowed
  - NOT suitable for most practical routing tools (too restrictive)
- Finds the optimal solution (# tracks = d)
- Nets are sorted according to their left endpoints
- Algorithm:

  Create an initial track t

  For all nets n_i in the order of their left endpoints

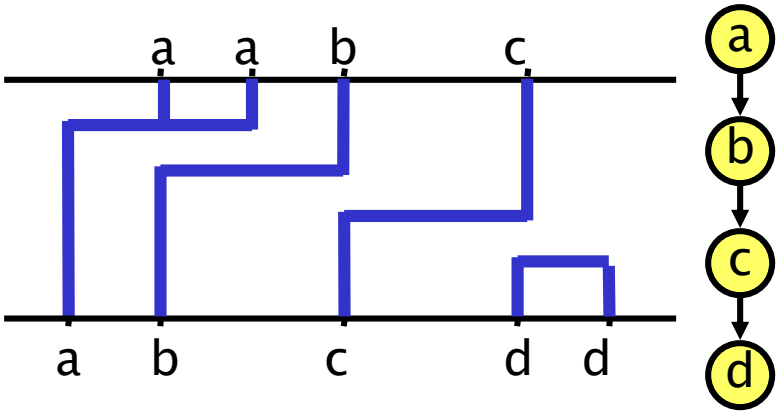
  if feasible to place the net on an existing track t_j,

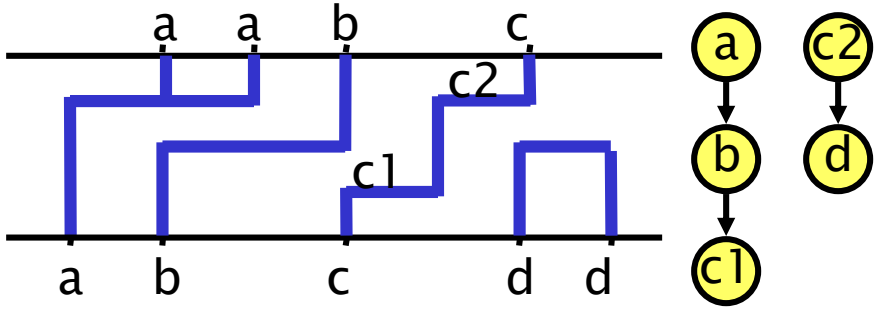
  assign net n_i to track t_j.

  else create a new track t_{new} and assign n_i to it.
- Time complexity: O(n log n)
- Routing example: see handout

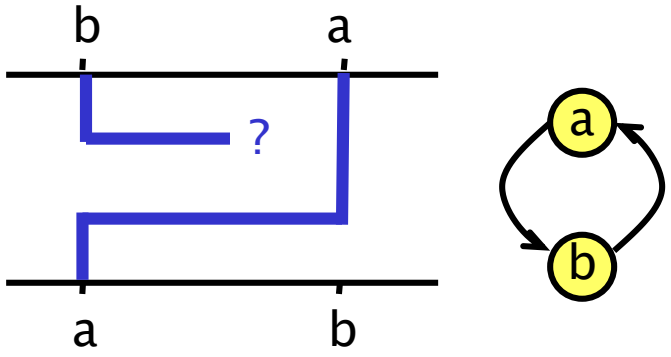
Doglegs

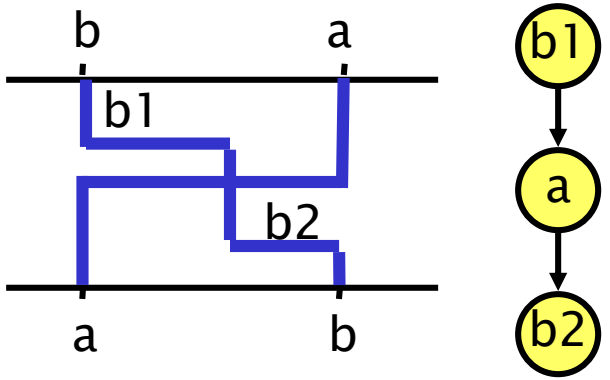
- Doglegs can reduce the longest path in VCG





- Doglegs break cycles in VCG

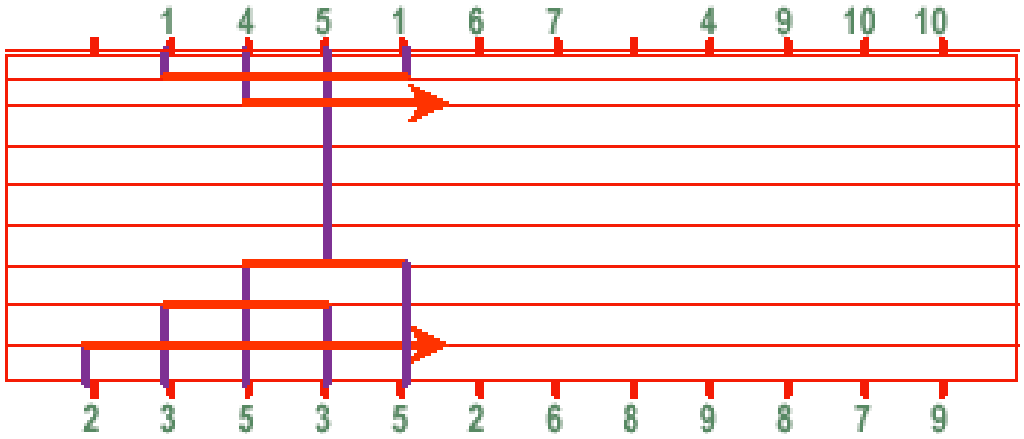




Greedy Channel Router

- Many greedy algorithms for channel routing exist
- Example: Rivest and Fiduccia DAC'82
  - Simple, linear algorithm
  - Guarantees routing of all nets
  - Uses doglegs (both restricted and unrestricted)
  - BUT may extend to right hand side of the channel
- Other techniques
  - Hierarchical: divide the channel into two smaller channels, route each small channel, merge
  - VCG reduction: nets that can be placed on the same track merged into one VCG node to reduce VCG size

Greedy Router: Rivest and Fiduccia

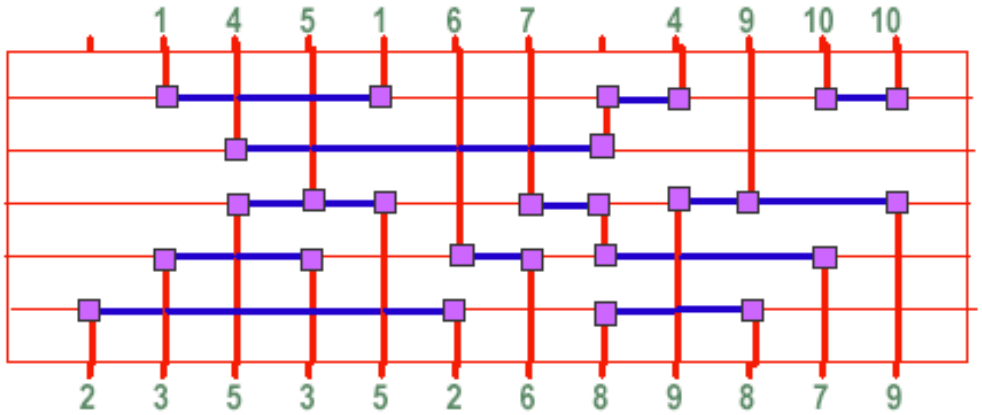


- Proceed column by column (left to right)
- Make connections to all pins in that column
- Free up tracks by collapsing as many tracks as possible to collapse nets
- Shrink range of rows occupied by a net by using doglegs
- If a pin cannot enter a channel, add a track
- O(pins) time

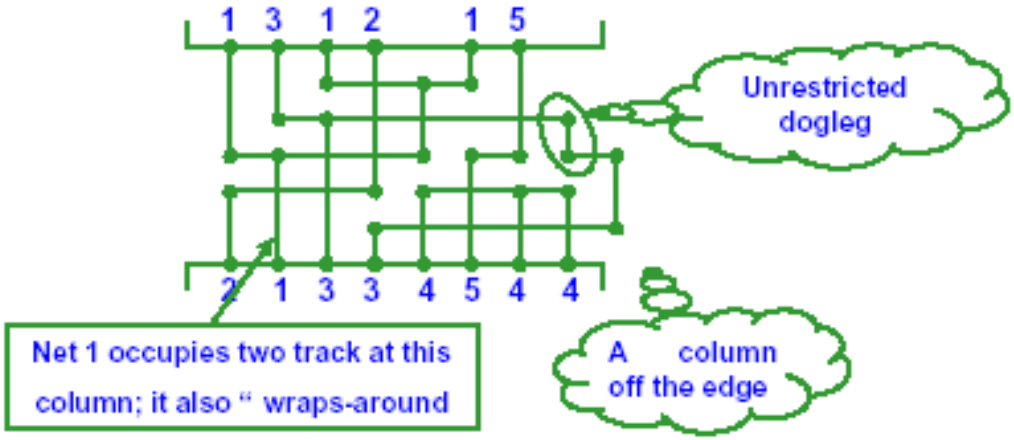
[©Keutzer]

Rivest and Fiduccia: Example

- Example output:



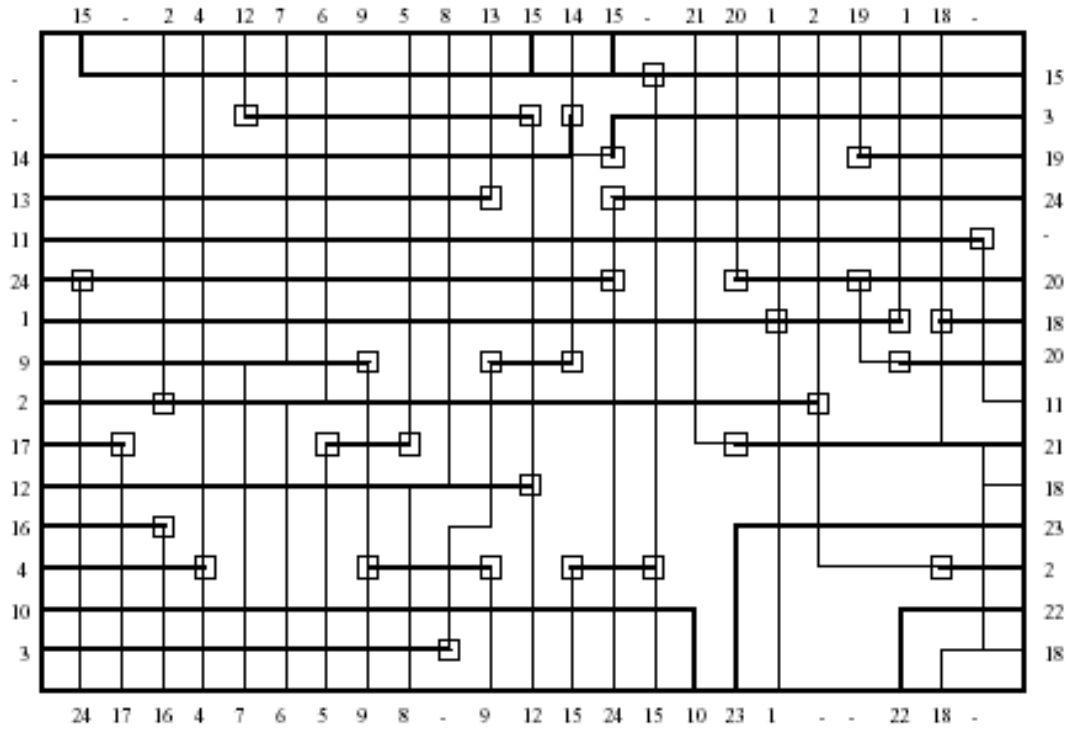
- Observations:
  - Always succeeds (even if cyclic conflict is present)
  - Allows unrestricted doglegs
  - Allows a net to occupy more than 1 track at a given column
  - May use a few columns off the edge



[©Keutzer]

Switchbox Routing

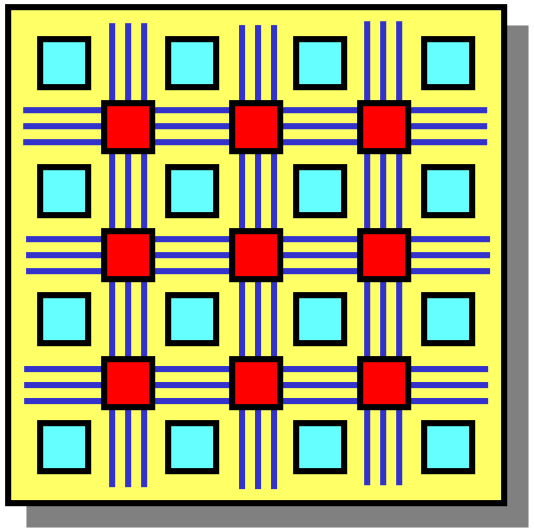
- Much harder problem than channel routing
  - Two-dimensional problem (channel routing was in a way one dimensional)
  - Need to solve in a hierarchical flow (split a channel into two, route one first, and route the second as a switchbox)
- A number of complex heuristic algorithms exist

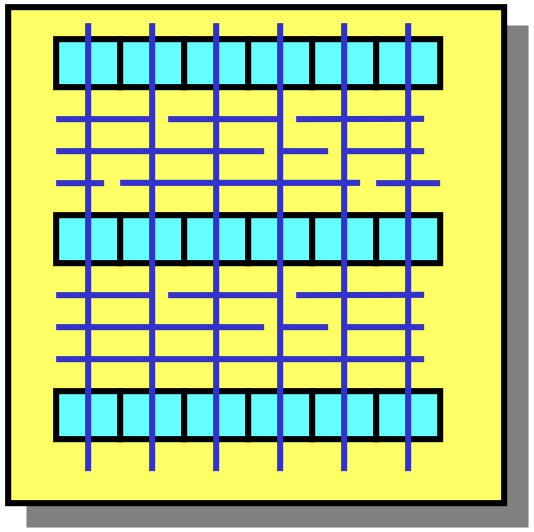


[©Sarrafzadeh]

FPGA Architecture - Layout

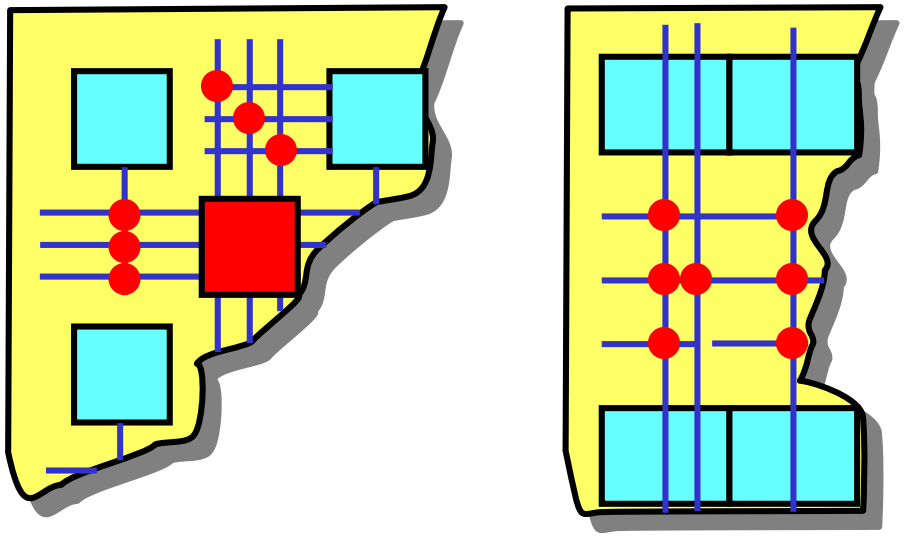
- Island FPGAs
  - Array of functional units
  - Horizontal and vertical routing channels connecting the functional units
  - Versatile switch boxes
  - Example: Xilinx, Altera
- Row-based FPGAs
  - Like standard cell design
  - Rows of logic blocks
  - Routing channels (fixed width) between rows of logic
  - Example: Actel FPGAs



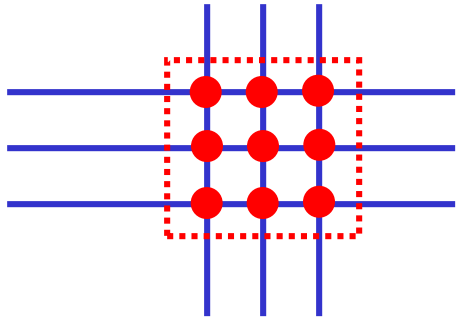


FPGA Programmable Switch Elements

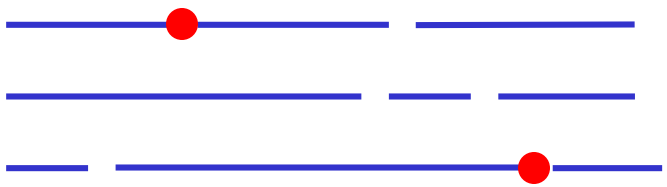
- Used in connecting:
  - The I/O of functional units to the wires



  - A horizontal wire to a vertical wire

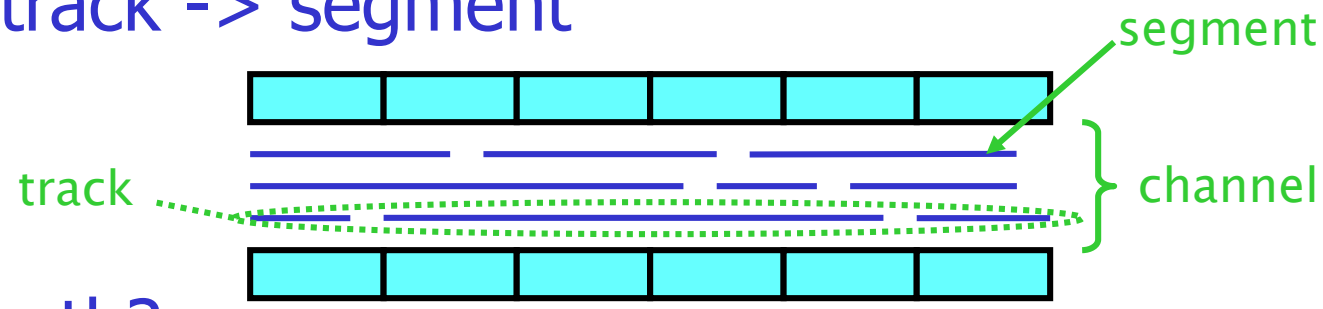


  - Two wire segments to form a longer wire segment

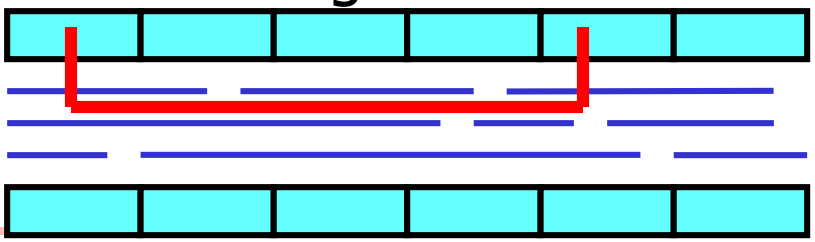


FPGA Routing Channels Architecture

- Note: fixed channel widths (tracks)
- Should "predict" all possible connectivity requirements when designing the FPGA chip
- Channel -> track -> segment

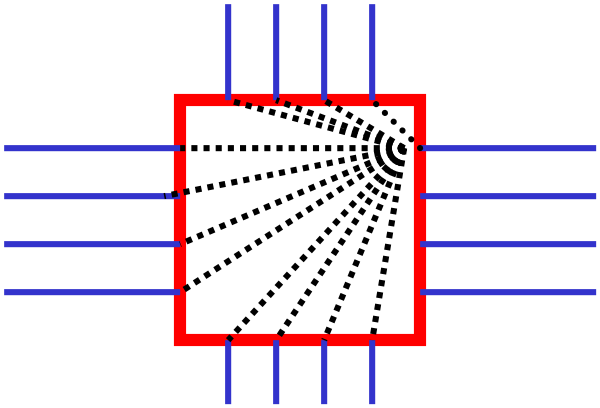


- Segment length?
  - Long: carry the signal longer, less "concatenation" switches, but might waste track
  - Short: local connections, slow for longer connections

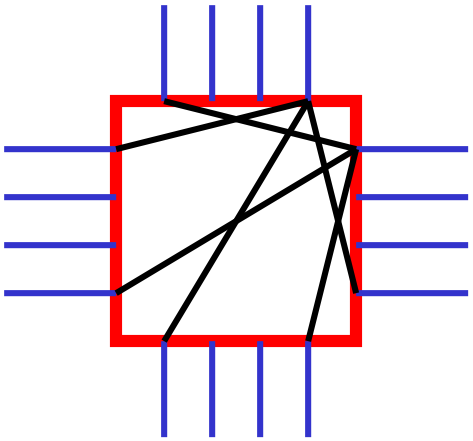


FPGA Switch Boxes

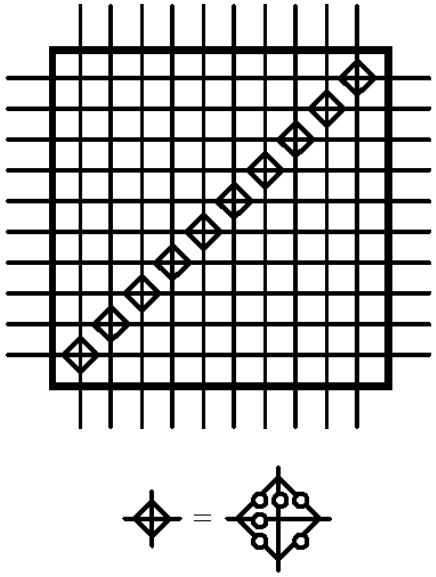
- Ideally, provide switches for all possible connections



- Trade-off:
  - Too many switches:
    - Large area
    - Complex to program
  - Too few switches:
    - Cannot route signals



One possible solution



Xilinx 4000

FPGA Routing

- Routing resources pre-fabricated
  - 100% routability using existing channels
  - If fail to route all nets, redo placement
- FPGA architectural issues
  - Careful balance between number of logic blocks and routing resources (100% logic area utilization?)
  - Designing flexible switchboxes and channels (conflicts with high clock speeds)
- FPGA routing algorithms
  - Graph search algorithms
    - Convert the wire segments to graph nodes, and switch elements to edges
  - Bin packing heuristics (nets as objects, tracks as bins)
  - Combination of maze routing and graph search algorithms